PATENT: UTILITY
Docket No: N0003/7030
Inventors: Linden A. deCarmo

METHOD AND APPARATUS FOR DYNAMICALLY BALANCING CALL FLOW WORKLOADS IN A TELECOMMUNICATIONS SYSTEM

METHOD AND APPARATUS FOR DYNAMICALLY BALANCING CALL FLOW WORKLOADS IN A TELECOMMUNICATIONS SYSTEM

RELATED APPLICATIONS

This application claims priority to U.S. Provisional Patent Application Serial No. 60/114,751, filed January 5, 1999, and entitled " SCALABLE CALL FLOW APPARATUS THAT DYNAMICALLY BALANCES WORKLOADS" by Linden A. deCarmo.

In addition, this application incorporates by this reference the subject matter of a U.S. utility patent application entitled " METHOD FOR DESIGNING OBJECT-ORIENTED TABLE DRIVEN STATE MACHINES", ~~Attorney Docket No. N0003/7020~~ 09/477,435, issued as US Patent No. 6,463,565 7/10/03, by Keith C. Kelly, Mark Pietras and Michael Kelly, commonly assigned and filed on an even date herewith.

FIELD OF THE INVENTION

This invention relates, generally, to telecommunication systems, and, more specifically, to a technique for managing call flows within a telecommunications system.

BACKGROUND OF THE INVENTION

Two fundamentally different switching technologies exist that enable communications. The first type, circuit-switched networks, operate by establishing a dedicated connection or circuit between two points, similar to public switched telephone networks(PSTN). A telephone call causes a circuit to be established from the originating phone through the local switching office across trunk lines, to a remote switching office and finally to the intended destination telephone. While such circuit is in place, the call is guaranteed a data path for digitized or analog voice signals regardless of other network activity. The second type, packet-switched networks, typically connect computers and establish an asynchronous "virtual" channel between two points. In a packet-switched network, data, such as a voice signal, is divided into small pieces called packets which are then multiplexed onto high capacity connections for transmission. Network hardware delivers packets to specific destinations where the

packets are reassembled into the original data set. With packet-switched networks, multiple communications among different computers can proceed concurrently with the network connections shared by different pairs of computers concurrently communicating. Packet-switched networks are, however, sensitive to network capacity. If the network becomes overloaded, there is no guarantee that data will be timely delivered. Despite this drawback, packet-switched networks have become quite popular, particularly as part of the Internet and Intranets, due to their cost effectiveness and performance.

In a packet-switched data network one or more common network protocols hide the technological differences between individual portions of the network, making interconnection between portions of the network independent of the underlying hardware and/or software. A popular network protocol, the Transmission Control Protocol/Internet Protocol (TCP/IP) is utilized by the Internet and Intranets. Intranets are private networks such as Local Area Networks (LANs) and Wide Area Networks (WAN). The TCP/IP protocol utilizes universal addressing as well as a software protocol to map the universal addresses into low level machine addresses. For purposes of this discussion, networks which adhere to the TCP/IP protocol will be referred to hereinafter "IP-based" or as utilizing "IP addresses" or "Internet Protocol address".

It is desirable for communications originating from an IP-based network to terminate at equipment in a PSTN network, and vice versa, or for calls which originate and terminate on a PSTN network to utilize a packet-switched data network as an interim communication medium. Problems arise, however, when a user on an IP-based or other packet switched data network tries to establish a communication link beyond the perimeter of the network, due to the disparity in addressing techniques among other differences used by the two types of networks.

To address the problems of network disparity, telecommunication gateways have been developed to allow calls originating from an IP-based network to terminate at equipment in a PSTN network, and vice versa, or for calls which originate and terminate on a PSTN network to utilize a packet-switched data network as an interim

communication medium. Gateway, such as the NetSpeak Model Nos. WGX-MD/24, a 24-port digital T-1 IP telephony gateway, or WGX-M/16, a 16-port analog IP telephony gateway, both commercially available from NetSpeak Corporation, Boca Raton, Florida, have a plurality of ports through which calls are handled.

Unlike traditional Public Branch Exchanges (PBXs), which merely processed the establishment of a call from one location to another, current telecommunication systems are expected to provide many types of optional services, such as call forwarding, call messaging, call waiting, and data entry, all transparently to the caller. In order to process these various functions, the gateways must be able to process the voice data stream and the call events associated with the call. Call events comprise any action related to a call, e. g. off-hook, on-hook, etc. However, it is desirable for gateway architectures to remain relatively rudimentary, performing only the handling of the data stream. Processing of the call events may be handled by a special server, referred to hereafter as a call flow server. In this manner the telecommunication systems may be updated to handle new types of call events by updating only the call flow server, instead of multiple gateways. Accordingly, gateways forward call events associated with a particular data stream to the call flow server and receive instructions from the call flow server as to how to handle or direct the data stream representing a call.

The call flow server uses algorithms known as "call flows" to handle one or more call events. A call flow typically comprises a series of instructions that control how one or more call events are processed. Such call flows are typically written in state tables, but may also be written in JAVA or any other type of computing language proprietary or not. Call flows are state machine operations that are managed on threads executing on a processor. However, the assignment of call flows to threads can cause problems.

In one technique, all call flow scripts are processed on a single thread. This solution is optimal for a single processor environment. However, this solution is not scalable as additional processing resources are added (i.e. the extra processors are ignored). In addition, a processor intensive call flow will block all other call flows from running (i.e. it is single tasking). In another technique, each call flow script is processed on a separate thread. This technique fully utilizes processor resources on multi-

processor machines and ensures that a script is never blocked because another script is running. However, it has the following disadvantages: 1) excessive context switches dramatically degrade performance on single processor machines; 2) a single thread per call flow is not realistic for large call flow environments that may process tens of thousands of calls simultaneously; and 3) call flows cannot be spread among multiple threads since one must ensure that events are received in the order they were sent and this cannot be guaranteed across threads.

Accordingly, there is a need for a method and apparatus that can adjust the call flow load within a single processor or multi-processor environment such that processing of threads associated with the call flows is optimized.

There is a further need for a method and apparatus for a flexible thread manager that has the performance of the single-threaded solution on a single processor system, but which scales intelligently when processors are added.

SUMMARY OF THE INVENTION

According to the present invention, a call flow server is disclosed that processes call flow events from a plurality of gateways bridging between traditional circuit-switched networks and packet-switched networks. The call flow server server, which may be implemented with either a single processor or multi-processor design, includes call flow engine and call flow thread manager modules capable of managing a plurality of call flow events by distributing the call flow scripts associated with such events among a plurality of threads executing on the call flow server. Each call flow event in the form of a call flow script is processed on a single thread within a selected processor. Processing each call flow script on a single thread fully utilizes the processor resources and ensures that a call flow script need not be blocked while another call flow script is running. The call flow server includes a thread manager to direct a given call flow script to a thread that has excess capacity.

According to one aspect of the present invention, a method is disclosed for distributing the call flow events among the plurality of threads executing within a telecommunications server. This method is performed to increase call flow event

processing efficiency and comprise the steps of: determining a call flow workload level for each of the plurality of threads; determining whether one of the plurality of threads is inefficiently handling its assigned call flow workload; and assigning call flow events from the inefficient thread to a second thread with excess call flow event handling capacity. The method may be further refined to include the steps of processing the call flow events within each of the plurality of threads or repeating selected steps until a balanced call flow event processing level is attained among the active threads.

According to another aspect of the present, a computer program product for use with a computer system may be implemented that includes program code for implementing the method steps described above. The computer program product may be distributed in the form of a computer useable medium such as a floppy disk, a CD-ROM disk, pre-installed on a hard disk storage drive of the communications server, or any other type of medium used to store data or program code for loading within a computer system, or , alternatively transmitted or propagated as part of a computer usable signal.

According to yet another aspect of the present invention, in a computer system, an apparatus for distributing call flow events among a plurality of threads, each thread having an associated call flow event queue in which call flow events queued, the apparatus comprises: a call flow engine configured execute call flow events associated with one of the threads; a call flow manager configured to distribute a plurality of call flow events among a plurality of threads used for managing the processing of plurality of call flows, n the call flow manager optimizing the processing of the call flows by determining which plurality of threads are operating inefficiently and reassigning a portion of the call flow events assigned to the inefficient thread to other of the plurality of threads having excess call flow processing capacity.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of a computer system suitable for use with the present invention;

Fig. 2 is a conceptual illustration of a communications network environment in which the present invention may be utilized;

Fig. 3 is a schematic diagram of a call flow server server in accordance with the present invention;

Figs. 4A-B illustrate a schematic diagram of call flow queues, threads, and the reallocation of call flow events from one thread to another in accordance with the present invention; and

Fig. 5 is a flow chart depicting the method for allocating thread resources in accordance with the present invention.

DETAILED DESCRIPTION

Fig. 1 illustrates the system architecture for a computer system 100, such as an IBM PS/2® computer on which the invention can be implemented. The exemplary computer system of Fig. 1 is for descriptive purposes only. Although the description below may refer to terms commonly used in describing particular computer systems, such as an IBM PS/2 computer, the description and concepts equally apply to other systems, including systems having architectures dissimilar to Fig. 1.

The computer system 100 includes a central processing unit (CPU) 105, which may include a conventional microprocessor, a random access memory (RAM) 110 for temporary storage of information, and a read only memory (ROM) 115 for permanent storage of information. A memory controller 120 is provided for controlling system RAM 110. A bus controller 125 is provided for controlling bus 130, and an interrupt controller 135 is used for receiving and processing various interrupt signals from the other system components. Mass storage may be provided by diskette 142, CD ROM 147 or hard drive 152. Data and software may be exchanged with computer system 100 via removable media such as diskette 142 and CD ROM 147. Diskette 142 is insertable into diskette drive 141 which is, in turn, connected to bus 130 by a controller 140. Similarly, CD ROM 147 is insertable into CD ROM drive 146 which is connected to bus 130 by controller 145. Hard disk 152 is part of a fixed disk drive 151 which is connected to bus 130 by controller 150.

User input to computer system 100 may be provided by a number of devices. For example, a keyboard 156 and mouse 157 are connected to bus 130 by controller 155. An audio transducer 196, which may act as both a microphone and a speaker, is connected to bus 130 by audio controller 197, as illustrated. It will be obvious to those reasonably skilled in the art that other input devices such as a pen and/or tablet and a microphone for voice input may be connected to computer system 100 through bus 130 and an appropriate controller/software. DNA controller 160 is provided for performing direct memory access to system RAM 110. A visual display is generated by video controller 165 which controls video display 170. Computer system 100 also includes a communications adaptor 190 which allows the system to be interconnected to a local area network (LAN) or a wide area network (WAN), schematically illustrated by bus 191 and network 195.

Computer system 100 is generally controlled and coordinated by operating system software, such the OS/2® operating system, available from International Business Machines Corporation, Armonk, New York or Windows NT operating system, available from Microsoft Corporation, Redmond, WA. The operating system controls allocation of system resources and performs tasks such as process scheduling, memory management, and networking and I/O services, among other things.
The present invention is intended for use with a multitasking operating system, such as those described above which are capable of simultaneous multiple threads of execution. For purposes of this disclosure a thread can be thought of as a "program" having an instruction or sequence of instructions and a program counter dedicated to the thread. An operating system capable of executing multiple threads simultaneously, therefore, is capable of performing multiple programs simultaneously.

In the illustrative embodiment, a call flow server server in accordance with the present invention is implemented using object-oriented technology and an operating system which supports an execution of an object-oriented programs. For example, the inventive call flow server server may be implemented using the C++ language or as well as other object-oriented standards, including the COM specification and OLE 2.0

specification for MicroSoft Corporation, Redmond, WA, or, the Java programming environment from Sun Microsystems, Redwood, CA.

Telecommunication Environment

Fig. 2 illustrates a telecommunications environment in which the invention may be practiced such environment being for exemplary purposes only and not to be considered limiting. Network 200 of Fig. 2 illustrates a hybrid telecommunication environment including both a traditional public switched telephone network as well as Internet and Intranet networks and apparatus bridging between the two. The elements illustrated in Fig. 2 are to facilitate an understanding of the invention. Not every element illustrated in Fig. 2 or described herein is necessary for the implementation or the operation of the invention.

A pair of PSTN central offices 210A-B serve to operatively couple various terminating apparatus through either a circuit switched network or a packet switched network. Specifically, central offices 210A-B are interconnected by a toll network 260. Toll network 260 may be implemented as a traditional PSTN network including all of the physical elements including routers, trunk lines, fiber optic cables, etc. Connected to central office 210A is a traditional telephone terminating apparatus 214A-D and an Internet telephones 232A-D. Terminating apparatus 214A-D may be implemented with either a digital or analog telephone or any other apparatus capable of receiving a call such as modems, facsimile machines, etc., such apparatus being referred to collectively hereinafter as a terminating apparatus, whether the network actually terminates. Further, the PSTN network may be implemented as either an integrated services digital network (ISDN) or a plain old telephone service (POTS) network. The Internet telephony is conceptually illustrated as a telephone icon symbolizing the Internet telephone client application executing on a personal computer and interconnected to central office 210A via a modem 270A. Similarly, telephone 214C is connected to central office 210B and WebPhone 232C is connected to central office 210B via modem 270C. Central offices 210A-B are, in turn, operatively coupled to Internet 220 by ISP 250B and 250C, respectively. In addition, central office 210A is coupled to ISP250B by

gateway 218B. Similarly, central office 210B is connected to ISP 250C by gateway 218C. In addition, a telephone 214B and Internet telephone 232B, similar to telephone 214A and Internet telephone 232A, respectively, are interconnected to Internet 220 via PBX 212, gateway 218A and ISP 250A. In addition, a global server 252, coupled to the Internet 220, may be implemented as described in U.S. Patent Application No. 08/719,894, entitled Directory Server for Providing Dynamically Assigned Network Protocol Addresses, incorporated herein. A global server suitable for use as Global Server 252 is commercially available from NetSpeak Corporation in the form of a collection of intelligent software modules including connection server Part No. CSR1, information server, Model ISR1, and database server, Model DBSR1. Finally, Internet Service Providers (ISPs) 250A-D may comprise any number of currently commercially available Internet service providers such as America On Line, the IBM Global Network, Compuserve, etc. An Intranet implemented as LAN 275 is coupled to Internet 220 via ISP 250D and server 256. Server 256 may have the architecture as illustrated in Fig. 1 and functions as a proxy server for LAN 275 to which WebPhone 232E is connected via a LAN-based TCP/IP network connector 280. A plurality of Internet telephone 232F and 232E are coupled to LAN 275 via LAN connectors 280.

A call flow server 300 is coupled over a packet-switched network to gateways 218A-C, as illustrated in Fig. 2. As described in greater detail hereinafter, gateways 218A-C forward call events to call flow server 300 which uses a call flow engine to efficiently handle processing of all call events. The gateways, call flow server and WebPhone client applications may be implemented as set forth in greater detail hereinafter.

WebPhone Client

Any of Internet telephones 232A-C shown in the Figures, and referred to hereafter simply as WebPhone(s), WebPhone process or WebPhone client 232, may be implemented as described in U.S. Patent Application Serial No. 08/533,115 entitled "POINT-TO-POINT INTERNET PROTOCOL" by Glenn W. Hutton,-filed September 25, 1995, incorporated herein by reference. An Internet telephony application suitable for

use with the present invention is the WebPhone 1.0, 2.0 or 3.0, client software application commercially available from NetSpeak Corporation, Boca Raton, Florida. The WebPhone client comprises a collection of intelligent software modules which perform a broad range of Internet telephony functions. For the purpose of this disclosure, a "virtual" WebPhone client refers to the same functionality embodied in the WebPhone client application without a graphic user interface. Such virtual WebPhone client can be embedded into a gateway, automatic call distributor, call flow server server, or other apparatus which do not require extensive visual input/output from a user and may interact with any other WebPhone clients or servers adhering to the WebPhone protocol.

The WebPhone software applications may run on the computer system described with reference to Fig. 1, or a similar architecture whether implemented as a personal computer or dedicated server. In such an environment, the sound card 197 accompanying the computer system 100 of Fig. 1, may be an Media Control Interface (MCI) compliant sound card while communication controller 190 may be implemented through either an analog modem 270 or a LAN-based TCP/IP network connector 280 to enable Internet/Intranet connectivity.

The WebPhone clients, as well as any other apparatus having a virtual WebPhone embodied therein, each have their own unique E-mail address and adhere to the WebPhone Protocol and packet definitions, as extensively described in the previously referenced related U.S. patent applications. For the reader's benefit, short summary of a portion of the WebPhone Protocol is set forth to illustrate the interaction of WebPhone clients with each other and the connection/information server 252 when establishing a communication connection.

Each WebPhone client, may serve either as a calling party or a caller party, i.e. the party being called. The calling party transmits an on-line request packet to a connection/information server upon connection to an IP-based network, e.g. the Internet or an Intranet. The on-line request packet contains configuration and settings information, a unique E-mail address and a fixed or dynamically assigned IP address for the WebPhone client. The callee party, also a utilizing a WebPhone client, transmits

a similar on-line request packet containing its respective configuration and setting information, E-mail address and IP address to the same or a different connection server upon connection to an IP-based network. The calling party originates a call by locating the callee party in a directory associated with either its own WebPhone client or the connection/information server to which it is connected. The callee party may be identified by alias, E-mail address or key word search criteria. Once the E-mail address of the callee party is identified, the calling party's WebPhone forwards a request packet to the connection/information server, the request packet containing the callee party's E-mail address. The connection/information server uses the E-mail address in the received request packet to locate the last known IP address assigned to the callee party. The connection/information server then transmits to the calling party an information packet containing the IP address of the callee party. Upon receipt of the located IP address from the connection server, the calling party's WebPhone client initiates a direct point-to-point communication link with the callee party by sending a call packet directly to the IP address of the callee party. The callee party either accepts or rejects the call with appropriate response packets. If the call is accepted, a communication session is established directly between the caller and the callee, without intervention of the connection/information server. The above scenario describes establishment of a communication link which originates and terminates with clients on an IP-based network. To facilitate interaction with WebPhone clients, a virtual WebPhone is implemented in the gateways 218A-C, as described hereinafter.

Gateways 218A-C shown in the Figures, any of which is referred to hereafter simply as gateway 218 acts as a proxy device and includes voice processing hardware that bridges from an IP-based network to a PSTN network. The gateway 218 may be implemented with either a microprocessor based architecture or with dedicated digital signal processing logic and embedded software. A gateway suitable for use as gateway 218 with the present invention is either NetSpeak Model Nos. WGX-MD/24, a 24-port digital T-1 IP telephony gateway, or WGX-M/16, a 16-port analog IP telephony gateway, both commercially available from NetSpeak Corporation, Boca Raton, Florida. Gateway

218 may be implemented using a computer architecture similar to computer system 100 described with reference to Fig. 1.

In addition, gateway 218 comprises one or more voice cards , one or more compression/decompression (codec) cards , and a network interface. The voice card(s) provides a T-1 or analog connection to the PBX or central office or analog telephone lines which have a conventional telephony interface, for example, DID, ENM. The voice card application program interface enable the instance of gateway 218 to emulate a conventional telephone on a PBX or central office of a PSTN carrier. Multichannel audio compression and decompression is accessed by gateway 218 via application program interfaces on the respective sound cards and is processed by the appropriate audio codec. Any number of commercially available voice cards may be used to implement voice card(s) within gateway 218. Similarly, any number of commercially available audio codecs providing adequate audio quality may be utilized. Each instance of gateway 218 interfaces with the TCP/IP network through a series of ports which adhere to the WebPhone protocol. Gateway 218 interfaces with the T1 line of the PSTN network through the interfaces contained within the voice card(s).

One of the capabilities of the gateway 218 is to bridge between the PSTN and Internet/Intranet, and the Internet/Intranet and the PSTN. Gateway 218 virtualizes the PSTN call, making it appear as just another WebPhone client call. This virtual WebPhone process interfaces with ACD server 242 so that incoming PSTN calls can be routed to agent WebPhone processes with the tracking, distribution, and monitoring features of the ACD server 242. For incoming calls originating on a PSTN, gateway 218 provides to ACD server 242 information about incoming calls so that proper call routing can ensue, such information possibly comprising Caller ID (CLID), automatic number identification (ANI), DNIS, PBX trunk information, from the central office 210, or other information collected by voice response units. In a similar manner, gateway 218 virtualizes the PSTN call, and transmits event information associated with the call to call flow server 300. Such information may be transmitted in packetized form using, for example the WebPhone protocol, or another standard or protocol.

Call Flow Server Architecture

Figure 3 illustrates conceptually the system architecture which may be used as the call flow server 300 of Figure 2. As Call flow server 300 may be implemented to execute on a computer architecture similar to computer system 100, as described in Figure 1, and an operating system, such as Windows NT. Call flow server 300 comprises multiple software modules that collectably enable call processing and call handling, including call flow event processing and handling. Specifically, call flow server 300 comprises a call flow engine 316, a call flow thread manager 318, and a call flow queue 320. Optionally, an Internet telephony application 322, which may perform any telephony feature such as automatic call distribution, call waiting, call forwarding, call conferencing, caller identification, or any other telephony feature, in a manner similar to the WebPhone 232 application described previously, may be included. A server suitable for use as call flow server 300 with the present invention is NetSpeak Gate Keeper 2. 1 commercially available from NetSpeak Corporation, Boca Raton, Florida. Alternatively, the call flow server of the present invention may be integrated into a number a different telecommunications apparatus, including an H.323 Standard Gatekeeper, a Session Initiation Protocol (SIP) server or a Media Gateway Control Protocol (MGCP) call agent used in packetized cable commuincations. As illustrated in Fig.2, call flow server 300 may be coupled directly to gateways 218B-C through a LAN or other network. Alternatively, call flow server 300 may be coupled to gateway 218A through the Internet, as illustrated.

The call flow engine 316 executes one or more call flow events, also known as call flow scripts, in order to process a call. A call flow event or script represents a state table or instruction(s) which the call flow event engine 316 executes. The call flow event state table calls functions that are provided with the script itself in a given script language, or in import libraries. Script language examples may include, but are not limited to, JAVA code, Object Oriented approaches in a language such as C++, or in any other proprietary script language. These functions can be in the form of "C" compiled library functions or script functions. If a new script begins execution at the request of a existing script, its state table takes effect.

Call flow engine 316 may execute multiple scripts concurrently. To be able to execute multiple scripts, call flow engine 316 utilizes multiple threads. At least one script executes per thread. To manage the number of scripts and to execute these multiple scripts concurrently, call flow engine 316 maintains instant information about each script concurrently executing.

Call flow scripts are ASCII based files that can be executed in an interpretive manner or compiled and executed. Call flow scripts have two components, the first is a state table while the second is a script function. The state table for a script defines the state events and their transitions. With each transition, a function or method is called. These script objects may be part of the script or they may be in an import library. A script object is made up of an event table and methods. A script object represents a single script state. Each object has a state of events they handle and are located within an event table. These events and methods are contained in an event table. A technique for designing object-oriented table driven state machines is disclosed in the previously referenced copending patent application, serial no. ~~XX/XXX,XXX~~ 09/477,435, entitled "METHOD FOR DESIGNING OBJECT-ORIENTED TABLE DRIVEN STATE MACHINES" by Keith C. Kelly, Mark Pietras and Michael Kelly, commonly assigned and filed on an even date herewith.

As stated previously, call flow server 300 may be implemented to perform any telephony function such as automatic call distribution, call waiting, call forwarding, call conferencing, caller identification, etc. A further detailed description of the complete call flow server 300, including the actual data associated tables and scripts which call flow engine 316 to function as a state machine are beyond the scope of this invention and will not be set forth herein for brevity.

Call flow thread manager 318 interacts with call flow engine 316 to manage the multiple threads handling call flow events within call flow server 300. Call flow thread manager 318 distributes call flow events among the respective call flow queues associated 320. Each thread has its own call flow queue 320 which is used to store a call flow script associated with a particular event. Optionally, an additional event queue, closely coupled, with the call flow script queue may be implemented. In such a

configuration, each event in the event queue contains a reference to a call flow script stored either in a table or the call flow script queue. A thread is defined as an execution path having at least one call flow instruction. Further, a thread has an associated context, which is the volatile data associated with the execution of the thread. A thread's context includes the contents of system registers and the virtual address space belonging to the thread's process. Thus, it is important to minimize thread context switches when readjusting thread call flow event handling efficiency.

Figure 4A and 4B illustrates conceptually a first thread 410 and a second thread 412 within a call flow server 300 executing on either a single processor or multiple processors. Associated with each of threads 410 and 412 is a call flow queue 320 loaded with one or more call flow events. The call flow queue associated with thread 410 includes call flows 414 and 416. The call flow queue associated with thread includes call flow 418. During operation, thread 410 may experience some type of processing delay where call flow 414 is unable to be processed promptly, thus preventing call flow 416 from being processed. Reasons for processing delay may include a heavy number of events being generated by call flow execution or heavy CPU processing by a given script. In the meanwhile, thread 412 has only call flow 418 to processed in its associated call flow queue. In order to maximize efficiency of call flow server 300, the call flow thread manager 318 evaluates the two threads 410 and 412 and their associated call flow queues to determine whether a call flow event reallocation should be performed in order to optimize call flow handling by the multiple threads, as described with reference to Fig. 5. Should such an event transfer occur, the results are shown in Figure 4B where call flow 416 has been transferred from first thread 410 to second thread 412.

Call flow thread manager 318 is configured to handle a number of threads scaling from a single thread on a single processor system to multiple threads for multiple processor systems. Furthermore, call flow thread manager 318 provides dynamic backlog detection. Specifically, if a call flow is not receiving enough processor resources, it is removed from the backlogged worker thread and added to a different thread as was shown in Figures 4A and 4B. Furthermore, call flow thread manager 318

provides intelligent call flow allocation. Call flow manager 318 allocates call flows based on the processor availability and processor work load. As a result, call flows are always allocated to the processor having the least amount of call flow load. Call flow thread manager 318 also minimizes context switches by arranging multiple call flows to run on the same thread where context is a factor in the thread processing.

Figure 5 is a flowchart of the process steps performed by call flow thread manager 318 to manage a plurality of threads within the telecommunications server, in accordance with the present invention. After starting in step 500, call flow thread manager 318 allocates the minimum number of worker thread objects for each thread, stores the queue depth and number of client call flows for each thread, as illustrated by step 510. During step 510, several constants and variables are initialized, including **MAX_ THREADS, MIN_ THREADS, MAX_ LOAD, MAX_ CALL_ FLOWS,** and **LOAD_CHECK_FREQUENCY**. **MAX_ THREADS** defines the maximum number of threads to allocate to the service call flows. **MIN_ THREADS** is the minimum number of threads to allocate to the service call flows. Typically, **MIN_ THREADS** is typically equal to the number of processors in the system or the number of threads that can be run by a single processor in a single processor system. **MAX_ LOAD** defines the maximum event queue depth for a worker thread. The event queue depth measures the delay experienced on a given thread when serving events for a given call flow. **MAX_ CALL_ FLOWS** define the maximum number of scripts that may be allocated to a thread. The call flow thread manager 318 prevents any thread from processing more than the max number of call flows defined by **MAX_ CALL_ FLOWS**, however, under heavy load conditions, this quantity may be exceeded, as necessary. The **LOAD_CHECK_FREQUENCY** controls the frequency that the event queue size is checked and is adjusted on various performance reasons such as minimum acceptable delay for processing a call flow or based on the number of threads that are actually available for processing call flows. To perform the above described initialization process call flow event manager 318 may execute the pseudo-code example set forth below:

static initialization (once per run):
allocate **MIN_THREADS** number of worker thread objects
for each thread, store queue depth and number of client call flows

script constructor:
call attachToThread()
attachToThread:

set the following variables:

```
minscripts = MAX_SCRIPTS_PER_THREAD;
minload    = HEAVILY_LOADED_QUEUE;

call findBestThread() to get the optimal thread for this call flow

// if minscripts == MAX_SCRIPTS_PER_THREAD or
        minload == HEAVILY_LOADED_QUEUE )
if ( there's no room in the current threads )
{
        if ( the total # of worker threads < MAX_THREADS)
                Allocate a new worker thread
                store queue depth and number of client call flows (0)
        else    // fit the call flow into a fully loaded queue
                // find the thread with the fewest scripts and least call
                // flow backlog
                // tell findBestThread to return ANY thread even if all are
                 //loaded it won't do this normally
                findBestThread( );

}
else
{
        attach this script to the worker thread
}
```

After the initialization has been performed in step 510, the call flow manager proceeds to step 512 to determine the number of available threads. After determining the total number of available threads, the call flow manager proceeds to step 514 where it allocates call flow events to the available threads within call flow server 300. Once call flow allocation has been performed, call flow event manager 318 determines the activity on each thread within call flow server 300, as illustrated by step 516.

Once the call flow thread manager 318 determines the activity on each thread, it determines whether any one thread or more has exceeded its maximum call flow capacity, as illustrated by step 518. To perform such a determination, call flow event manager 318 may execute the pseudo-code example set forth below:

```
workersMaxedOut: quick check to see if all threads are at capacity

set maxedOut = true

if ( active worker threads equals MAX_THREADS)
    {

      loop through all the threads
     {
       grab the queue size of the thread

            if ( the queue size is less than the max permitted backlog AND total # of client
            call flows < MAX_SCRIPTS_PER_THREAD )
     {
                    // this thread still has capacity for more client
                     //call  flows
                    maxedOut = false;
        exit loop
      }

     }


    }
    else
    {
            maxedOut = false;
    }

return maxedOut;
```

If no thread has exceeded its given call flow capacity, thread manager 318 returns to monitor the thread activity. If a thread has exceeded its call flow capacity, call flow thread manager 318 allocates the excess call flow load to another thread, as illustrated by step 520. The criteria used to allocate thread call flow load from one

thread to another typically includes determining the thread having the fewest scripts and the least call flow backlog as well as the thread that has the greatest amount of resources available for use. The call flow thread manager 318 locates the thread having the greatest resources available and allocates the blocked scripts to that particular thread. To determine which thread has the greatest resources, call flow event manager 318 may execute the pseudo-code example set forth below:

findBestThread: searches for the thread with the lightest load

```
while ( there are more threads to search through)
{
        grab a description of the load of the current thread
        if ( this thread is running fewer call flows than the max # acceptable to the caller)
                grab a snap shot of the event queue size

        if ( the event queue is smaller than the max event queue size permitted by the caller)
        {
                indicate that this script is the smallest amount we've seen so far
                if this thread has no clients and no backlog, exit loop since we've found a free
                thread!
        }

}
```

Otherwise, the system selects a first available thread having adequate resources for processing.

In step 522, call flow thread manager 318 determines whether a call flow balance has been achieved among the plurality of threads. If such balance has been achieved, then the call flow thread manager has performed its task and returns. If a proper balance has not been achieved, then the call flow thread manager 319 returns to step 520 to allocate call flow events among the plurality of threads until a balance is achieved. Balance is achieved when no thread exceeds **MAX_LOAD**.

Once the scripts have been allocated to their various threads, they are added or stored in the call flow queue associated with that thread. To add a call flow event to a

call flow queue, call flow event manager 318 may execute the pseudo-code example set forth below:

addElement: add an event to a call flow's event queue
each call flow actually shares a queue with all of the other call flows on that thread

Increment checksize
Increment # of outstandingEvents

// we check to see if the threads should be load balanced every >
LOAD_CHECK_FREQUENCY events.

```
if ( loadcheck > LOAD_CHECK_FREQUENCY )
    {
        queueSize = size of thread event queue

        // if the queue is heavily loaded AND our instance
        // isn't reponsible for this load AND
            // there's another thread with capacity.....
            if ( queueSize > HEAVILY_LOADED_QUEUE &&
                !( eventsOutstanding > ( queueSize >> 2 ) ) &&
                !workersMaxedOut() )
            {
                        remove all this call flows events from the event queue and store in a temp
                        variable.

                remove this call flow from this worker thread ....
                    //pick the best available thread.....
                attachToThread();

                                // transfer our events to the new thread's //queue....

            }
        }

            add the requested event to the queue

    }
```

Once in call flow queue 320, the scripts are processed by the call flow engine 316 until such time as all the call flow events have been processed. Each worker

thread may execute the pseudo-code example set forth below to effect processing of call flows:

eventProcessed: reduces # of events that still must be processed...
Increment # of outstandingEvents

serviceEvents: pulls an event from the queue and sends it to the appropriate call flow

```
for every event in the queue
                {
                  retrieve the first element in the queue
                  remove first element from the queue

                  invoke call flow method to handle the event

                  // let client know we've processed the event...
                  eventProcessed();


                }
```

run: main thread worker routine

```
loop forever
sleep till a call flow generates an event
serviceEvent()
```

The reader will appreciate that the inventive algorithm described herein has the following advantages: 1) a configurable number of threads, that is, it is scalable from a single thread on a single processor systems to multiple threads for multiprocessor systems; 2) dynamic backlog detection, e.g. if a call flow is not receiving enough processor resources, it is removed from the backlogged worker thread and added to a different thread; 3) the algorithm is lightweight and almost as fast as the single processor approach; 4) call flows are allocated based on processor availability and processor workload enabling call flows to be allocated to the processor with the least

load; and 5) context switches are minimized since multiple call flows can run on the same thread.

It is important to distinguish at this time that call flow events and the scripts which they are written are state events that are processed by the computer system within the telecommunications server. Call flow events are not the actual data stream of information being transmitted from one user to another in the form of either audio, video, or other type of file transfer information. Call flow events are actions that are typically requested by one of the client applications or endpoints or by the server itself. These actions typically can include call transactions such as call waiting, call forwarding, call messaging, billing for a particular client, and any other call action that is intended to be secondary to the actual calling information being carried over the call servicing network of Figure 2. It is intended that where possible, these call flow events are processed in a manner that is reasonably transparent to the overlying purpose of the phone connection.

A software implementation of the above-described embodiments may comprise a series of computer instructions either fixed on a tangible medium, such as a computer readable media, e.g. diskette 142, CD-ROM 147, ROM 115, or fixed disk 152 of Fig. 1, or transmittable to a computer system, via a modem or other interface device, such as communications adapter 190 connected to the network 195 over a medium 191. Medium 191 can be either a tangible medium, including but not limited to optical or analog communications lines, or may be implemented with wireless techniques, including but not limited to microwave, infrared or other transmission techniques. The series of computer instructions embodies all or part of the functionality previously described herein with respect to the invention. Those skilled in the art will appreciate that such computer instructions can be written in a number of programming languages for use with many computer architectures or operating systems. Further, such instructions may be stored using any memory technology, present or future, including, but not limited to, semiconductor, magnetic, optical or other memory devices, or transmitted using any communications technology, present or future, including but not limited to optical, infrared, microwave, or other transmission technologies. It is

contemplated that such a computer program product may be distributed as a removable media with accompanying printed or electronic documentation, e.g., shrink wrapped software, preloaded with a computer system, e.g., on system ROM or fixed disk, or distributed from a server or electronic bulletin board over a network, e.g., the Internet or World Wide Web.

Although various exemplary embodiments of the invention have been disclosed, it will be apparent to those skilled in the art that various changes and modifications can be made which will achieve some of the advantages of the invention without departing from the spirit and scope of the invention. Further, many of the system components described herein such as the client application and the gateway have been described using products from NetSpeak Corporation. It will be obvious to those reasonably skilled in the art that other components performing the same functions may be suitably substituted. Further, the methods of the invention may be achieved in either all software implementations, using the appropriate processor instructions, or in hybrid implementations which utilize a combination of hardware logic and software logic to achieve the same results. Such modifications to the inventive concept are intended to be covered by the appended claims.

What is claimed is: